Appl. No. 10/051,535
Amdt. Dated December 27, 2005
Reply to Office action of August 4, 2005

RECEIVED
CENTRAL FAX CENTER
DEC 2 7 2005

IN THE SPECIFICATION

Please amend the Specification as follows:

Please amend all the paragraphs starting on page 13 line 10 and ending on page 15 line 6, as follows:

Figure 3A illustrates the overall structure of an exemplary UDM Design Vector 200 in accordance with one embodiment of the present invention. The exemplary UDM Design Vector [[200]] 100 can contain Run Method 210 and a definition of the "Matched" execution engine hardware, referred to as a Conjugate Virtual Machine (CVM) [[215]] 115. Hence, the UDM Design Vector 200 can be represented as a software module executing on its own virtual (hardware) processor referred to as CVM. Header [[205]] 105 and Trailer [[220]] 120 contain the binding methods that connect the Design Vector [[200]] 100 to other Design Vectors. Run Method [[210]] 110 generally contains the behavior of the Design Vector [[200]] 100.

In one embodiment, the Run Method [[210]] 110 can include a Java software module that describes the processing to be performed by the UDM Design Vector [[200]] 100. Temporal [[225]] 125 typically contains the invocation method of the Design Vector. CVM [[215]] 125 generally contains the description of the execution engine, which can be the JVM instruction subset that is needed to support the execution of the Run Method [[210]] 110.

In one embodiment, the Header [[205]] 105 generally contains the description of the input variables and the UDM Design Vectors that produce the input variables. In this embodiment, the Trailer [[220]] 120 generally contains the description of the output variables and the UDM Design Vectors destined to receive the output variables.

In addition, the UDM Design Vectors [[200]] 100 use the following types of Header and Trailer variables, including Local Variables, Shared Variables, and Global

Variables. Local Variables are generally local within an UDM Design Vector. Shared Variables are typically shared between two Functional Design Vectors. Global Variables can be shared between several Functional Design Vectors.

Shared Variables and Global Variables can be accessed by multiple Design Vectors, and hence can be used to pass data between Design Vectors. Shared Variables and Global Variables are defined in the Header [[205]] 105 and Trailer [[220]] 120 of the Design Vector [[200]] 100. Synchronization of access to Shared & Global Variables is performed data synchronization mechanisms provided by the selected design language. Examples of data synchronization mechanisms can include "wait()" and "notify()" methods, as defined by the Java programming language.

Temporal information [[225]] 125 contains the invocation information and the maximum allowable response time within which the Design Vector [[200]] 100 must complete its processing.

Conjugate Virtual Machine (CVM) field [[215]] 125 generally includes design information describing required computing capabilities or features of the Scaled Virtual Machine that are minimally sufficient to execute the sequence of operations described in the Method field [[210]] 110. Later in the Unified Design Methodology 100 (shown in Figure 2A), the realization of the Scaled Virtual Machine as described in the CVM field [[215]] 125 can be committed to either hardware or software depending upon the specified capabilities of a preferred platform [[150]] (shown in Figure 2A). In one embodiment, the programming language can correspond to the default CVM that is being used. For example, the Java programming language would be used if the default virtual machine were a JVM. In this embodiment, the CVM describes the hardware that is the matched subset of the JVM instruction set generated by compiling the Design Vector Run Method.

Please amend all the paragraphs starting on page 16 line 7 and ending on page 18 line 10, as follows:

Figure 3B shows an exemplary set of Design Vector Attributes [[230]] <u>130</u>, including Vector Name [[235]] <u>135</u>, Vector Type [[240]] <u>140</u>, and Parent Application Syntax [[245]] <u>145</u>. Vector Name [[235]] <u>135</u> usually is the same as Class Name. Vector Type [[240]] <u>140</u> can be used to indicate whether the vector is a Functional Vector or an Interconnect Vector. Parent Application Syntax name [[245]] <u>145</u> is generally the Name of the Parent Vector.

Figure 3C shows exemplary variables definitions [[250]] <u>150</u> including Header [[255]] <u>155</u> and Trailer [[260]] <u>160</u> binding information. In one embodiment, the Header binding information [[255]] <u>155</u> can include definitions of input variables and the name of the source vector generating these input variables. In this embodiment, the Trailer binding information [[260]] <u>160</u> can include definitions of output variables and the name of the destination vector that will absorb these output variables.

Figure 3D shows exemplary Methods [[270]] <u>170</u> of an exemplary UDM Design Vector, including a Vector constructor method [[272]] <u>172</u>, a vectorRun() method [[274]] <u>174</u>, a vectorInvocation() method [[276]] <u>176</u>, a getHeaderInput() method [[278]] <u>178</u>, a sendTrailerOutput() method [[280]] <u>180</u>, a run() method [[282]] <u>182</u>, and other Java specific methods used to complete the vector -- such as initialize(), wrapup(), vectorGet(), vectorSend(), vectorWait(), headerDataReady(), trailerDataReady().

The Vector constructor method [[272]] <u>172</u> is generally called when the vector is first created. When called, the Vector constructor method [[272]] <u>172</u> stores the Vector Attributes [[230]] <u>130</u> (shown in Figure 3B) and receives the Header and Trailer binding information [[255]] <u>155</u>, [[260]] <u>160</u> (shown in Figure 3C).

The vectorRun() method [[274]] 174 can generally be invoked to perform the vector function. The vectorInvocation() method [[276]] 176 generally contains the invocation and temporal information and waits until these requirements are satisfied. The getHeaderInput() method [[278]] 178 can be used to obtain the Header binding information [[255]] 155 (shown in Figure 3C). The sendTrailerOutput() method [[280]] 180 can be used to send the trailer variables to the bound vector that consumes the Trailer. The run() method [[282]] 182 should exist in each Java thread and should be executed automatically when the Java thread is created.

Two exemplary types of UDM Design Vectors in accordance with one embodiment of the present invention were shown on Figure 1A. The UDM Design Vectors are the most basic building blocks with which the terminal architectural model can be constructed. As shown in Figure 1A, the two exemplary types of UDM vectors include Functional Vectors [[286]] 186 and Interconnect Vectors [[284]] 184. In one embodiment, each Functional Vector [[286]] 186 can be dedicated to performing a single function of transforming the input variables into the output variables as described in the run() method [[282]] 182 (shown in Figure 3D). It should be noted that Functional Vectors [[286]] 186 could be restricted to receive their input variables from a single Interconnect Vector and deliver their output variables to a single Interconnect Vector.

In one embodiment, Interconnect Vectors [[284]] 184 can be used to provide the interconnectivity between Functional Vectors [[286]] 186. Each Interconnect Vector [[284]] 184 can typically perform the functions required to contain and transport shared and global variables between Functional Vectors [[286]] 186. If an Interconnect Vector [[284]] 184 interconnects two Functional Vectors [[286]] 186, variables exchanged between the two Functional Vectors [[286]] 186 are shared variables. If an interconnect vector transports variables between more than two Functional Vectors [[286]] 186, the transported variables are global variables.

As previously stated, Figure 1B shows an exemplary Terminal Structural model 290 that is expressed in terms of a set of interconnected Functional Vectors [[292]] <u>192</u> in accordance with one embodiment of the present invention. Interconnect Vectors [[294]] <u>194</u> are used to link or interconnect the Functional Vectors [[292]] <u>192</u>. As previously stated, Interconnect Vectors [[294]] <u>194</u> and Functional Vectors [[292]] <u>192</u> represent the lowest level of building blocks that will be used in building the terminal structural model.